Artificial Intelligence Project---RLE and MIT Computation Center

Writing and Debugging Programs---Memo 6

by S. Russell

Open subroutines

A subroutine is a fixed set of instructions that is used many times. The kind most often used explicitly are closed subroutines such as MAPLIST which are set up so that they may be used by any part of a program. Open subroutines are ones which are inserted wherever they are necessary in a program, examples of these are car and cdr.

The program:

```
        LXD JLOC,4
        CLA 0,4
        PAX 0,4
        PXD 0,4
```

will compute car(J). Since this set of instructions is so short it would waste a significant amount of time to make it a closed subroutine.

Other simple open subroutines presently in the LISP language are:

```
cdr(J):   LXD JLOC,4
          CLA 0,4
          PDX 0,4
          PXD 0,4

cwr(J)    LXD JLOC,4
          CLA 0,4

dec(J):   CLA JLOC
          ANA DECM     where DECM has 1'0 in its decrement only

add(J):   CLA JLOC
          PAX 0,4
          PXD 0,4
```

* Note that car(J)=add(cwr(J)).

The program made by joining cwr and add is simplified to get cdr.

Similarly, when a string of open subroutines is joined together redundant operations can usually be eliminated.

Closed subroutines in LISP are given a standard calling sequence. The arguments of the subroutines are placed in order in the AC, the MQ, and locations called ARG3 and ARG4.

After the arguments have been provided, the subroutine is entered by a TSX SUBR,4 where SUBR is the location of the first instruction of the subroutine. The subroutine ends with its value in the AC and the instruction TRA 1,4 which returns control to the instruction after the TSX that set IR4. Note that any other program can be executed between this TSX and the TRA 1,4 so long as the contents of IR4 are restored to the value set by the TSX before the TRA is executed. A subroutine restores all index registers it uses to the value they had on entering the subroutine.

When a 15-bit quantity is an argument or value, it is always stored in the decrement of a word; a 1-bit quantity is stored in the low order bit of the decrement.

ORDER OF COMPUTATION:

In general, functions are calculated from the inside out. For example, car(cdr(J)) cdr(J) is calculated first; then car of this intermediate result is calculated to get the final result.

In multi-argument functions the first argument is calculated first and so on. For example, in compiling J=cons(car(J),cdr(J)) the order of calculation would be:

car(J); cdr(J); cons(car(J),cdr(J));

Temporary storage must, of course, be provided to save car(J) while cdr(J) is being calculated. A 704 program to calculate

This would be:

```
LXD JLOC,4
CLA 0,4                 CWR(J)
PAX 0,4                 CAR(J)
SXD T1,4
LXD JLOC,4
CLA 0,4                 CWR(J)
PDX 0,4                 CDR(J)
SXD T2,4                C
LDQ T2                  CDR(J)
CLA T1                  CAR(J)
TSX O$CONS,4            CONS(CAR(J),CDR(J))
STO JLØC                J*
```

In some cases it is possible to rearrange the order of computation to give a more efficient program, but in other cases this may lead to unpleasant side effects, especially if the rearrangement effects input-output routines. The above can be made faster without ill effects by calculating cdr(J) first:

```
LXD JLØC,4
CLA 0,4                 CWR(J)
PDX 0,4                 CDR(J)
SXD T2,4
LXD JLOC,4              CWR(J)
CLA 0,4                 CAR(J)
PAX 0,4                 CAR(J)
PXD 0,4
LDQ T2                  CDR(J)
TSX O$CØNS,4            CONS(CAR(J),CDR(J)
STØ JLØC
```

The resulting 10% saving in time is almost worth the effort. (The additional confusion does not justify the change at all).

SAVE, UNSAVE AND RECURSIVE FUNCTIONS

Consider the "slow" maplist in Memo 4: maplist(L,f)=(L=0⟶0, 1⟶cons(f(L), maplist (cdr(L),f)))

The order of computation should be

is L=0?
possible return
f(L)
cdr(L)
maplist(cdr(L),f)
cons(cdr(L),maplist(f(L),f))
return

Temporary storage is needed to hold the result of calculating f(L) while cdr(L) and maplist are being calculated. If we made no provision for saving the value of this temporary storage location while maplist was being calculated the recursion would leave a different value in this temporary storage location. Similarly, index register 4 must be saved. To avoid this, we use SAVE and UNSAVE in every routine that could possibly end up using itself. This includes not only recursive routines, but also routines that can have an arbitrary function as an argument, since the arbitrary function might be ,or include ,the routine itself.

The use of SAVE and UNSAVE is explained in Memo 4 p1-2. Only temporary quantities that are used both before and after the self reference(s) need be saved. The value of IR4 is quantity of this type.

A good example of the use of SAVE and UNSAVE is the program for copy(L) on p.10 of memo 4.

Since the function will not use itself if either of the first two conditions are satisfied, SAVE and UNSAVE need not be used in either of these cases. Note that IR4 is stored in CS1 long before the use of save. CT1 used for temporary storage of L, but this is only needed before the reference to copy so it is not saved. Since SAVE and UNSAVE each take 14+3N instructions to save N registers, they should not be used needlessly.

DEBUGGING PROGRAMS

To make sure a program works, it must be tested on some sample problems. These problems must include all the special cases that occur in possible input to the program, all the special cases that the program must consider (these 2 sets are not identical) and some typical cases. The test program itself must present the routine with these test problems through the proper calling sequence. It may be helpful to write the test program before the main program, so that the test program is not biased by knowledge of how the main program works.

The test program must be simple and yet give as much information as possible. Since few programs work the first time,post-mortem requests should be sprinkled through the test program to allow a bug to be pinned down quickly. Print early and often. Remember, you can always ignore what is printed if the program works. Don't have senseless large dumps, because they tie up the off-line printer; it only prints 2 pages a minute. The best compromise seems to start with trivial test problems and print out the results and several key registers. (For example, the public push down list, the free storage list and places where IR4 has been stored). After the routine works on trivial problems, complex ones and obscure special cases can be tried.

It is extremely important that in the final tests the routine tested is in one block and complete, so that it can be used by putting that block into another program without any modifications.

THE DEBUGGING PACKAGE

To make debugging simple we have prepared a basic package for the MIT automatic operator system containing error post-mortems,

final post-mortems, setup of free storage and public push down lists, and most of the debugged subroutines. To use this, it is only necessary to write the test program that starts at the first location and ends with TRA DØNE. There are some symbols that cannot be defined in the test program and the main program because they are used in this package, but these are mostly function names. With the package deck the programmer need only provide the test program to be is inserted in the deck, put a RUN card at the beginning of the deck, and a TER card at the end. The RUN and TER cards should have the same title, which should contain the problem number, the programmer's number, and a distinctive symbol, all separated by minus signs. Before running a program the programmer should be familiar with SAP, described in CC-82, the MIT post-mortem program MIPMR1, described in CC-58, and the MIT automatic operator system described in CC-108. Copies of these CC-memos can be gotten from the Computation Center office.

After the entire program and tester have been written, they should be reread in detail, to check for errors both large and small. Doubly defined and undefined symbols are one of the most common errors encountered at this point.

Due to an idiosyncracy of the off-line card reader, program cards cannot have a 9-punch in colum 73. This excludes the letters I, R, and Z.

When your program has been run, you will get two listings: an on-line one, and an off-line one. Except for times and on-line output, the off-line listing has everything on it the on-line does.

In the extreme left of a sap listing, letters such as U,M,A,D, occasionally appear. These are explained in CC-82 and they signal a trouble of some kind. Not all of these troubles

cause a program to be tagged as bad, so they should be looked for even if the program was good.

If there is a number other than 0 in the "4 FAIL" column of the "SHARE ASSEMBLER STATISTICS" there was an error in reading a tape, and the program may be rerun.

Do not be confused by the fact that printing done by your program occurs in operator phase 2, while the post-mortem output occurs in phase 3.

CS-TR Scanning Project
Document Control Form

Date : 11 / 30 / 95

Report # AIM - 6

Each of the following should be identified by a checkmark:
Originating Department:

- [x] Artificial Intellegence Laboratory (AI)
- [ ] Laboratory for Computer Science (LCS)

Document Type:

- [ ] Technical Report (TR)
- [x] Technical Memo (TM)
- [ ] Other:

Document Information

Number of pages: 7(11-IMAGES)
Not to include DOD forms, printer intstructions, etc... original pages only.

Originals are:
- [x] Single-sided or
- [ ] Double-sided

Intended to be printed as :
- [x] Single-sided or
- [ ] Double-sided

Print type:
- [ ] Typewriter
- [ ] Offset Press
- [ ] Laser Print
- [ ] InkJet Printer
- [ ] Unknown
- [x] Other: COPY FROM MIMEOGRAPH MATERIAL

Check each if included with document:
- [ ] DOD Form
- [ ] Funding Agent Form
- [ ] Cover Page
- [ ] Spine
- [ ] Printers Notes
- [ ] Photo negatives
- [ ] Other:

Page Data:

Blank Pages(by page number):

Photographs/Tonal Material (by page number):

Other (note description/page number):

Description :	Page Number:
IMAGE MAP! (1-7)	1-7
	(8-11) SCANCONTROL, TRGT'S(3)

Scanning Agent Signoff:

Date Received: 11 / 30 / 95 Date Scanned: 12 / 14 / 95 Date Returned: 12 / 14 / 95

Scanning Agent Signature: Michael W. Cook

Rev 9/94 DS/LCS Document Control Form cstrform.vsd

Scanning Agent Identification Target

Scanning of this document was supported in part by the **Corporation for National Research Initiatives,** using funds from the **Advanced Research Projects Agency** of the **United states Government** under Grant: **MDA972-92-J1029.**

The scanning agent for this project was the **Document Services** department of the **M.I.T Libraries.** Technical support for this project was also provided by the **M.I.T. Laboratory for Computer Sciences.**